SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 10, No. 1, February 2013, 199-208

UDK: 004.383/.384 DOI: 10.2298/SJEE1301199S

Power Management Implementation in FreeRTOS on LM3S3748

Mirela Simonović[1], Lazar Saranovac[1]

Abstract: Power consumption has become a major concern of embedded systems today. With the aim to reduce power consumption during the runtime, operating systems are dealing with power management. In this work, the FreeRTOS port is extended with power management features on LM3S3748 microcontroller. Tickless idle technique is implemented to provide more power-saving during the processor idle periods.

Keywords: Power management, Low power, Embedded systems, RTOS, Operating systems.

1 Introduction

In embedded systems green technology implies the development of hardware and software solutions that reduce energy consumption. In general, efficient power management in embedded systems reduces the cost of providing power to the chip and lowers the chip temperature. In battery-operated devices, less energy consumption increases battery lifetime.

Because power consumption has become a major concern of embedded systems, many low power techniques were introduced in the manufacture of hardware platforms. Power consumption can be lowered by gating clocks to the hardware modules that are not currently used. Dynamic power dissipation of CMOS logic is proportional to the operating frequency and to the square of the operating voltage. Therefore, lowering clock frequency and voltage supply level during the runtime can be used for power saving when top performance of a system or module is not required.

To reduce the power consumption during the runtime, operating systems provide power management features. Most of the real time operating systems do not have well designed power management, or do not provide support for low power modes usage at all. Because of the hardware platform dependency, FreeRTOS, just as many other operating systems, does not provide power management features. This work is focused on extending the FreeRTOS port for the LM3Sxxxx family of microcontrollers with power management. The goal is

[1]School of Electrical Engineering, University of Belgrade, P.O. Box 35-54, 11000 Belgrade, Serbia, E-mails: mirelasimonovic@gmail.com, laza@el.etf.rs

to put the microcontroller into the low power mode, when there is no useful work for the processor to be done. To keep the track of time, FreeRTOS waste power when the processor is idle, because the system remains in the Run mode. To deal with this problem, a power management technique known as *tickless idle* has been developed. The *tickless idle* technique implies the turning off the system timer from the idle task function, so as to enter sleep mode and ensure that the processor is operating only when there is a ready task to execute or generated interrupt needs handling.

The Software is developed in IAR Embedded Workbench for ARM. Extended FreeRTOS port is tested on EK-LM3S3748 evaluation board [1], that contains LM3S3748 microcontroller based on the ARM Cortex-M3 processor [2].

2 LM3S3748 Low Power Modes

The Cortex-M3 provides Sleep and Deep Sleep modes as a power management feature [3, 4]. In both modes, the processor clock is gated and therefore code is no longer executed. The clock frequency of active peripherals is unchanged in the Sleep mode, but in the Deep Sleep mode the clock frequency is lowered. For additional power savings in Deep Sleep, the on-chip LDO voltage regulator can be programmed to adjust the output voltage level to lower values. During the Deep Sleep mode, PLL is turned off. Sleep mode should be entered when the processor is idle and the system clock frequency cannot be lowered, or when the lowest possible wake-up latency is required. For additional power savings, the Deep Sleep mode can be entered. A drawback of the Deep Sleep compared to the Sleep mode usage is increased wake-up latency.

The low power mode can be invoked by executing either WFI (Wait For Interrupt) or WFE (Wait For Event) instruction. If the low power mode is entered by executing the WFI instruction, only enabled interrupts can wake up the processor. After a wake up from the WFI power saving, the execution continues from the interrupt service routine. The WFE provides a mechanism for conditional entering into the low power mode. If the event register is 0, the WFE execution invokes the low power mode. Otherwise, WFE resets the event register and behaves as a NOP instruction. If the SEVONPEND bit in the system control register is set, every interrupt transition from inactive to pending state sets the event register [4, 5]. After a wake-up from the WFE power saving, the execution continues from the interrupt service routine if the interrupt that caused the wake-up is enabled. Otherwise, the execution continues after a WFE call. Whether the Sleep or the Deep Sleep mode is entered, it is determined by the SLEEPDEEP bit in the system control register, regardless of which instruction invoked the low power mode.

3 FreeRTOS

FreeRTOS is a free and open source real time operating system designed to have small footprint and targeted to embedded systems [6]. It is written in C language and does not contain drivers, support for complex memory management or networking. Scheduling algorithm is simple round-robin with priorities, also a co-operative, preemptive or hybrid scheduling is configurable. Both tasks and coroutines are supported, but in this work only tasks are considered. Tasks can be blocked for a specified time, when they are called delayed. Blocking tasks for a specified time is usually used for creating periodic tasks. If tasks are blocked indefinitely, they are called suspended. Tasks can be unsuspended by calling an appropriate function from the interrupt service routine or by some other task.

FreeRTOS keeps the track of time by counting periodically generated interrupts as ticks. In the official FreeRTOS ports for microcontrollers based on the Cortex-M3, the Systick timer is used as a tick source. Each time the Systick interrupt routine executes, an internal FreeRTOS variable called *xTickCount* is incremented and a check is performed whether any delayed task has to be deblocked. All internal time-based calculations, such as task delay time, depend on the xTickCount value. For the correct functioning of FreeRTOS, no Systick interrupt should be neglected, because deadlines of some tasks could be missed.

The Cortex-M3 has a powerful possibility of disabling all interrupts that are below or equal to a priority determined by the BASEPRI register. Those interrupts are called low priority interrupts. Interrupts whose priority is greater than the BASEPRI register value are called high priority interrupts. As FreeRTOS uses this mechanism for disabling interrupts, they are never all disabled. By the system's-kernel function, only low priority interrupts can be disabled. In the interrupt service routines of high priority interrupts, the usage of the kernel system functions is not allowed. Therefore, the kernel function for disabling interrupts is used for keeping critical sections safe from other tasks preemption. This mechanism's advantage is that handling high priority interrupts is never delayed by the kernel system code.

4 Tickless Idle Implementation

The only FreeRTOS system task is the Idle task, which has the lowest priority. In FreeRTOS, the Idle task not only executes the infinite loop to feed the processor with instructions, it deletes tasks which have finished their execution and also may contain some application hook functions that are usually used for runtime statistics. The low power mode could be entered in the Idle task function when Idle finishes the usual job, unless there are some user tasks ready to execute. Even when the Idle task is being executed, there might be user tasks ready to execute. For example, while the Idle task was running, some external event caused a user task to be unsuspended. If the preemptive kernel is not configured, the unsuspended task will continue execution when the current timeslice expires or the Idle task yields. Consequently, before entering the low power mode a check for ready tasks must be performed and if there are any, the low power mode must not be entered.

If the Sleep mode was entered by invoking the WFE or WFI instruction, the Systick would wake up the processor periodically every time a tick is generated. In some special use-cases, this realization could be acceptable, therefore it is also implemented. In most cases, this is an unreasonable power wasting, because the processor wakes up unnecessarily and often. Also, in the Deep Sleep mode, because of the changed system clock frequency, the Systick would be slower in generating ticks and would give wrong information to the system about the elapsed time. In order to obtain more power-saving, a *tickless idle* solution is implemented. The *tickless idle* implies disabling the tick source from the Idle task function, in order to ensure that the processor remains in the low power mode for longer periods of time. When the Systick is disabled, it is necessary to tackle the problem of time tracking in the low power mode.

For that purpose a timer is used. Generally speaking, it would be suitable to use a timer whose clock source is independent from the system clock, having in mind that the system clock frequency changes in the Deep Sleep mode. One solution, as developed in [7], would be to use the RTC timer. Using the RTC timer for the Sleep mode time tracking also ensures that the processor stays longer in the Sleep mode, than when timer clocks have higher frequency. However, due to hardware bugs in LM3S3748 [8], the RTC timer cannot be used. Instead, a 32-bit general purpose timer clocked with the system clock was used in this work. At the moment of entering the low power mode, it is known when a user task has to continue execution. That information is obtained from the delayed task list and is used for the configuring of the timer to generate interrupt and wake up processor just before the task has to be deblocked. If there are no delayed tasks, the processor can remain in the Sleep mode for $2^{32}–1$ system clock cycles.

When the system resumes from the low power mode, the *xTickCount* variable has to be updated in accordance with the slept ticks count, before the schedule of tasks resumes. Updating xTickCount is complicated, because some asynchronous interrupt can cause a wake-up. Therefore, after resume from low power mode, it must be examined what was the wake-up source. If timer interrupt has woken-up processor, there is no need to calculate number of slept ticks. Only already known tick count that was used for timer configuring has to be added to *xTickCount* value. If some other interrupt caused wake-up, current timer value is read and accordingly *xTickCount* variable is updated. After general purpose timer is disabled and Systick is enabled, task scheduling resumes.

In regular FreeRTOS operating, the *xTickCount* variable overflows every 2^{32} tick. When such an overflow happens, the pointers to FreeRTOS internal lists have to be exchanged. Pointers to delayed task control blocks are kept in those lists. In [7], processor is woken up to handle pointers exchange before the overflow occures. There is no need to additionally wake up the processor for handling the pointers exchange. A better solution is implemented in this work: pointers are exchanged after the processor is woken up if the overflow was supposed to happen while the processor was in the low power mode.

5 Protecting Critical Section

Another problem to solve is caused by the fact that the process of entering the low power mode must be non-interruptable by other tasks. Supposing for example, that all the checks have been done and all the conditions for entering the low power mode have been met, the next thing to do is to appropriately configure the general purpose timer and to execute the WFI or WFE instruction. Suppose that after the configuring of the timer some interrupt is generated and in the interrupt service routine a task is unsuspended. If preemptive scheduling is configured, that would cause an unsuspended task to continue execution immediately. A saved program counter value of the Idle task will result upon entering the low power mode next time the Idle continues execution, without even performing checks if that has been allowed and also without properly configured general purpose timer. If the cooperative scheduling is configured, there will still be a problem: the unsuspended task will not execute immediatelly, but it will once the processor wakes up from the already configured low power mode. In both configurations a deadline would be missed and the system could crash.

In order to protect the critical section when entering the low power mode, there is no need to disable all interrupts. Only low priority interrupts, which in the interrupt service routine can cause a task to unsuspend, should be disabled. If low priority interrupts were disabled before the process of entering the low power mode, and the WFI instruction executed after that, only high priority interrupts could cause a wake-up. Therefore, the WFE instruction is used with the SEVONPEND bit set. When the SEVONPEND bit is set and the WFE is used for power saving, every interrupt transition from the inactive to the pending state is a wake-up event. This means that even a disabled low priority interrupt can cause a wake-up.

Entering the low power mode critical section is protected from other tasks preemption, by invoking the FreeRTOS system function for disabling interrupts.

Before the low priority interrupts are disabled, the event register should be reset. If any low priority interrupt occurs after the low priority interrupts are disabled, it will transit to the pending state and cause the event register to be set. In that case, the WFE execution behaves as the NOP instruction and the low power mode is not entered. When the *xTickCount* variable is updated and the Systick is enabled, low priority interrupts are enabled and a pending interrupt is served. If preemption after executing an interrupt service routine happens because a task was unsuspended, the saved program counter of the Idle task points out of the critical section, what is a desired behavioral.

If the Sleep mode is used, high priority interrupts are served without any delay. If a high priority interrupt occurs after the configuring of the system clock for the Deep Sleep mode and before the system clock is reconfigured for the Run mode, the interrupt service routine would be served with a slower clock rate. That is the overhead of using the Deep Sleep mode. Interrupt service routines of low priority interrupts always execute with high frequency clock configured for the Run mode, regardless of when an interrupt occurs.

6 Implemented Solutions

By configuring the system macros, the user choses which low power mode is to be used. If the Deep Sleep mode is configured, the generated kernel code takes more memory compared to the Sleep mode, as shown in **Table 1**.

Table 1
Generated code sizes and measurement results.

Mode	Code Size [B]	I_C [mA]
Run	6778	129.0
Sleep non-tickless	7000	80.0
Sleep tickless	7560	80.0
Deep Sleep tickless 1	7844	52.8
Deep Sleep tickless 2	7960	52.8

Two different solutions for Sleep mode usage are implemented. The WFI instruction is used for entering the Sleep mode in the non-tickless solution, because there is no critical section that has to be protected. The processor is woken up periodically by a Systick interrupt, so the *xTickCount* variable doesn't have to be additionally updated. This solution (Sleep non-tickless, **Table 1**) saves the least power compared to all tickless solutions. Another solution for the Sleep mode is tickless (Sleep tickless, **Table 1**). As the general purpose timer is configured to generate an interrupt, and also due to disabling and enabling the

Systick, a drift in time appears. Unfortunately, the drift has an accumulative effect on the xTickCount value and cannot be compensated. In order to configure the general purpose timer, the number of ticks for which microcontroller should remain in the Sleep mode has to be converted to an equivalent number of the system clock cycles.

For the Deep Sleep mode, the system clock is lowered to 32 kHz and can be more divided. Optionally, the on-chip LDO voltage regulator can be programmed to adjust a lower voltage level for the Deep Sleep mode. The user configures the value of the system clock frequency in the Deep Sleep mode, and has to pay attention to the consequences of the changing clock rate. First of all, high priority interrupts can occur when the clock frequency is low and the system would react slower. Every interrupt handling during the suspend-resume process causes more drift in time. As the general purpose timer does not have a clock source independent from the system clock, the system clock frequency is adjusted to 32 kHz, then the timer is configured. Consequently, before the WFE call, a part of critical section executes with a slow clock rate. Additional delay is entered after the system resume by configuring the PLL and setting the system clock for the Run mode, executing with a slow clock rate, too. When 32 kHz is configured for the Deep Sleep system clock frequency, the measuring has shown that the suspend-resume cycle takes approximately 30 ms, unless in that period of time high priority asynchronous interrupts are generated. That is why it must be checked in how many milliseconds a task will have to be deblocked. If the calculated time is less than 30 ms + x, Deep Sleep must not be entered. Value of x is configurable.

When configuring the general purpose timer that generates an interrupt while processor is in Deep Sleep, a conversion from tick count to equivalent low frequency clock cycles must be done. When calculating clock cycles that the timer has to count with a low frequency clock, rounding in calculation causes error. After the system resume, if the timer has not generated an interrupt, the current timer value is read and again calculations are made for the conversion to an equivalent tick count. All rounding errors affect the xTickCount value and have an accumulative effect. Deviation of the xTickCount value is proportional to the number of asynchronous interrupts that occur during the Deep Sleep, and also to the number of entries to the Deep Sleep mode. In order to prevent the accumulation of errors, two solutions for using the Deep Sleep mode are implemented. One is based on more energy saving: after an asynchronous interrupt causes a wake-up, another enter into Deep Sleep is tried (Deep Sleep tickless 2, **Table 1**). In another solution, when an asynchronous interrupt occurs, the processor remains in the Run mode, waiting for the general purpose timer to generate an interrupt (Deep Sleep tickless 1, **Table 1**). This solution provides less energy saving, but also decreases the error accumulation.

7 Measurements

Because board's voltage supply is constant, power consumption is proportional to current consumption. A current consumption is measured using the high side current sense monitor ZXCT1081 with a gain of 10 and a voltage output. Externally, a $R_{\mathrm{SENSE}} = 1\Omega$ resistor is added. The signal output is measured with the oscilloscope LeCroy WaveAce 222 (2×220MHz) and equals $10I_C R_{\mathrm{SENSE}}$, where I_C is drawn current. Measured current is drawn by EK-LM3S3748 board, not only by the microcontroller.

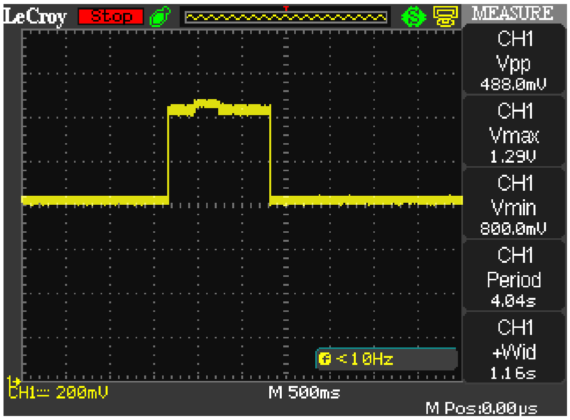

Fig. 1 – *Measurement result – Sleep mode.*

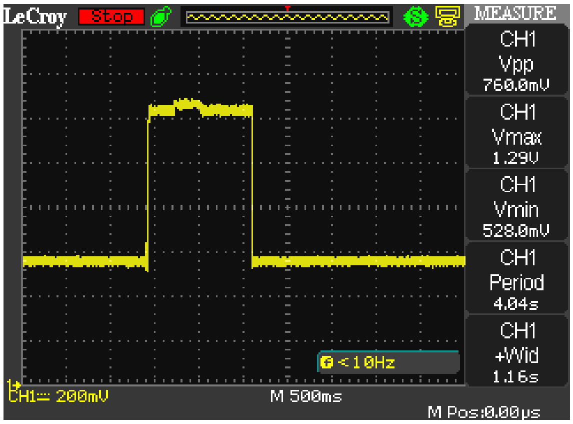

Fig. 2 – *Measurement result – Deep Sleep mode.*

In general, the energy consumption depends on the used peripherals and amount of time spent in each mode, which is application defined. For measuring, a simple application with two tasks that are toggling diodes and sending messages to the serial port is used. The same applications is executed with different configurations of FreeRTOS power management: Sleep (Fig. 1) or Deep Sleep mode (Fig. 2).

For tested application, the average current is shown in **Table 2**. If Sleep mode is entered during the processor's idle periods, the average current is lowered by 38.1 mA. More power saving is achieved using Deep Sleep, so that average current is lowered by 59.2 mA.

Table 2
Measured Average Current.

Configured Low Power Mode	Average I_C [mA]
-	129.0
Sleep tickless	90.9
Deep Sleep tickless 1, 2	69.8

Because in measured use case asynchronous interrupts are not generated, the average currents for Deep Sleep tickless 1 and 2 solutions are equal.

8 Conclusion

In this work, the FreeRTOS port is extended to support low power modes usage for the LM3S3748 microcontroller based on the Cortex-M3. Sleep and Deep Sleep modes are used, and in total four solutions are implemented. One solution of the Sleep mode is based on keeping the track of time in the Sleep mode as in the Run mode. Because of frequent and unnecessary waking-up, this solution provides the least energy saving compared to all other solutions. It is implemented because it does not involve any inaccuracies, no delays, which tickless solutions normally involve.

In order to induce even more power saving, the *tickless idle* power management technique is implemented. For keeping the track of time while the processor is in the low power mode, the general purpose timer is configured to generate an interrupt at an appropriate moment and wake up the processor. The general purpose timer was the only option for time tracking, because of hardware bugs in the used microcontroller. A disadvantage of using the general purpose timer is a drift in time that has an accumulative effect. The drift occurs due to the configuring of the timer, disabling and enabling the Systick. An additional drift is caused by changing the system clock frequency for the Deep Sleep mode. The best solution would be to use a timer that has a separate clock source and does not depend on the system clock frequency. For that purpose, on some another platform the RTC timer can be used, as it is used in [7, 9]. Even a better solution would be to make FreeRTOS totally tickless, because there would be no need for counting elapsed ticks during the low power state. Another improvement is related to the fact that in the implemented solutions the decision which low power mode should be entered is made before compile time. This approach is acceptable when all the tasks are created before the scheduling starts and are never deleted. In other situations, it would be much better if the decisions were brought dynamically during the runtime. The implemented solutions can be easily extended to support the decisions dynamically.

The measurements have shown that implemented solutions have an impact on reducing power consumption during the processor's idle periods.

9 Acknowledgment

This work was done as part of the research program on the master studies at School of Electrical Engineering.

10 References

[1] Texas Instruments: Stellaris LM3S3748 Evaluation Kit User's Manual, Jan. 2010.

[2] Texas Instruments: Stellaris LM3S3748 Microcontroller Data Sheet, Jan. 2011.

[3] J. Yiu: The Definitive Guide to the ARM Cortex-M3, 2nd Edition, Elsevier, Burlington, MA, USA, 2010.

[4] ARM: Cortex-M3 Devices Generic User Guide, Dec. 2010.

[5] ARM: ARM v7-M Architecture Reference Manual, Nov. 2010.

[6] R. Barry: Using the FreeRTOS Real Time Kernel – A Practical Guide, 2009.

[7] M. Tverdal: Operating System Direct Power Reduction on EFM32, Master Thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2010.

[8] Texas Instruments: Stellaris LM3S3748 RevA0 Errata, Aug. 2011.

[9] A. Spalluto: Energy Aware RTOS for EFM32, Master Thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2011.